European Journal of Academic Essays, Special Issue (1): 27-30, 2014
ISSN: 2183-1904
www.euroessays.org

Recent Trends in Analysis of Algorithms and Complexity Theory

Analysis of time complexity of algorithm (dependent and independent loops)

< Mishra Sambit Kumar >[1], < Champati Pratap Kumar >[2], < Sahoo Rajesh Kumar >[3]

[1] Associate Professor, Department of Computer Sc.& Engg.

Ajay Binay Institute of Technology, Cuttack

sambit_pr@rediffmail.com

[2] Sr. Lecturer, Department of Computer sc.&Engg.

Ajay Binay Institute of Technology, Cuttack

pkchampati@gmail.com

[3] Assistant Professor , Department of Computer Sc.& Engg.

Ajay Binay Institute of Technology, Cuttack

rajesh_sahoo@rediffmail.com

Abstract: The pragmatic aspects of computing require one to be cognizant of the resource usage aspects of a program. While such concerns should be secondary to those of the correctness of the program, correctness can make the difference between success and failure in computer problem solving. The general term used to resource usage is complexity.

The amount of time required to execute an algorithm is known as time complexity. Similarly, the amount of memory required to execute an algorithm is known as space complexity. There are several ways to measure time complexity,

(a) the average case, which is more to compute because it requires some knowledge that is expected on average and

(b) the best case, not exactly the best measure of an algorithm's performance because unless it is likely to continually be the best case comparisons between algorithms.

Keywords: Complexity, NP complete, Nested loop, Iteration, Dominant terms

1. Introduction

Efficiency of algorithms can be measured in terms of execution time and the amount of memory required. The time complexity is nothing but measurement of the amount of time required to execute an algorithm. Factors that should not affect time complexity analysis are (a) programming language chosen to implement an algorithm, (b) the quality of the compiler, (c) the speed of the computer on which the algorithm is to be executed.

Formally, the time complexity T(n)of an algorithm is O(f(n)) if, for some positive constants c1,and c2 for all finitely many values of n , c1*f(n)<=T(n)<=c2*f(n).
This gives upper and lower bounds on the amount of work done for all sufficiently large n.

The highest order term of the expression may be chosen $T(n)=n^2+n/2$, with a coefficient of 1, so that $f(n)=n^2$. The dominant term of T(n) may determine the basic shape of the function.

2. Review of Literature

E.horowitz et.al[1] have mentioned that a program may be written containing some simple commands and some timing commands. After that the duration of operations used in the algorithm may be found out. The weakness of this is that it is machine dependent, programming language dependent, compiler dependent etc. In the light of these goals, the primitive operations may be counted instead of timing them. All the operations may not be even counted. With these

Corresponding Author: Sambit Kumar Mishra

simplifications, the time complexity may be computed instead of running time.

Dewdney A.K et.al[2] have discussed in their survey that time complexity allows for comparisons with other algorithms to determine which is more efficient. A path may be needed to determine whether or not something is going to take a reasonable amount of time to run or not. They also discussed that an NP complete problem may be computed in polynomial time. It can also be verified in polynomial time.

J.E. Hopcroft et.al[3] have elaborated in their survey that if L can be decided by an s(n) space bounded turing machine, then L can also be decided by an s(n)/2 space bounded turing machine. Similarly if L can be decided by a t(n) time bounded turing machine, then L can also be decided by an n+t(n)/2 time bounded turing machine.

Graham et.al[4] have discussed in their paper that, given access to the code of a program, the complexity can be determined by analysis. However, it may be desirable to check the analysis empirically. A way to proceed experimentally is to run the programs on inputs of a variety of values of the input measure and record the time in each case. This may give a set of time pairs, which may constitute an approximation to the time complexity of the program.

D. Kempe et.al[5] have discussed in their paper about the objectives of time complexity analysis that focused to determine the feasibility of an algorithm by estimating an upper bound on the amount of work performed. Also they compared different algorithms before deciding which one to implement. They also suggested that time complexity analysis for an algorithm is independent of the programming language and the machine used.

M uhlenbeiи et.al[6] have discussed in their paper that the function OneMax is one of the best known test functions for randomized search heuristics. For several classes of heuristics, initial theoretical studies focused on the analysis of OneMax . Probably the first rigorous runtime analysis of an evolutionary algorithm was done on this function.

Droste et.al[7] presented a first rigorous runtime analysis of an estimation-of-distribution algorithm on OneMax and other linear functions. After that rigorous runtime analyses of an ant colony optimization algorithm on OneMax was implemented.
Rylander, B et.al[8] have discussed in their paper that, the complexity of an optimization problem for a genetic algorithm is bound above by the growth rate of the smallest representation , Minimum chromosome Length that can be used to solve the required problem. This is because the probabilistic convergence time will remain fixed as a function of the search space. All things held constant, the convergence time will grow as the search space grows. Also it was observed that the complexity of the training data has a direct correlation to the complexity of the desired output.

A. V. Aho et.al[9] have discussed in their paper about the efficiency of an algorithm that depends on its use of resources, such as: the time it takes the algorithm to execute, the memory it uses for its variables, the network traffic it generates and the number of disk accesses it makes.

3. Problem Formulation

3.1. Algorithm-1

With the independent nested loops, the number of iterations of the inner loops is independent of the number of iterations of the outer loop.

```
int t=0;

for(int i=1; i<=n/2;i++)

for( int j=1; j=n*n;j++)

t=t+i+j;
```

Outer loop executes n/2 times. For each of these times, inner loop executes n^2 times , so the body of the inner loop is executed $(n/2)*n^2 = n^3/2$ times. The algorithm is $O(n^3)$.

3.2. Algorithm-2

With dependent nested loops, the number of iterations of the inner loop depends on a value from the outer loop.

```
int t=0;

for ( int i=1; i<=n; i++)

for( int j=1; j<3*j;j++)

t=t+i;
```

When i is 1, inner loop executes 3 times; when i is 2, inner loop executes 3*2 times. When i is n, inner loop executes 3*n times. In all the inner loop executes $3*n^2/2 + 3*n/2$ times.

To determine the time complexity of an algorithm,

(a) the amount of work done may be expressed as a sum of f1(n)+f2(n)+----------ft(n). (b) the dominant term fi may be identified such that fj is O(fi) and for t different from j, ft(n) < fj(n) .

Examples of dominant terms :

n dominates log2(n).

n*log2(n) dominates n.

n^2 domintes n*log2(n).

n^j dominates n^t when j>t.

a^n dominates n^t for any a>1 and t>=0.

Algorithms whose solutions are independent of the size of the problem's input usually have constant time complexity. Constant time complexity is denoted as O(1). The time complexity of an operation may be recognized which modifies the data structure without a formal proof. Many operations on the data structures are clearly O(1), retrieving the size, testing emptiness etc.

4. Experimental analysis:

Consider an algorithm : Parameters: A positive integer, n.
Returns: The sum of the integers from 1 to n. { sum := 0;
for i := 1 upto n { sum := sum + i;
}
return sum;
}
The running time of the algorithm is the time taken to initialize sum and to return it, which will be the same no matter what n is, plus the time spent to execute the loop body, which may depend on the value of n. Suppose the first and last commands take 4 microseconds to execute in total, and suppose the assignment in the loop body takes 2 microseconds to execute, then the running time may be defined in microseconds of the algorithm for input n, t(n): t(n) =def 4 + 2n.

The running time may be expressed as a function of the input size.

Consider another example.

Parameters: A finite-length list, L, of positive integers.
Returns: The sum of the integers in the list.
{ sum := 0; for each x in L
{ sum := sum + x;
}
return sum;
}

Making similar assumptions, and simplifications, the running time of the above algorithm, t(n), might also be 4 + 2n, where n is the length of L. But it is clear that for any problem instance of size n , the time taken may be 4 + 2n microseconds.

4.1. Table : Inputs processed at fixed time intervals

Sl.No.	*Running time*	*1 Sec.*	*1 Min*
01	700n	4000	180,000
02	60n(logn)	9056	279,679
03	2*n**2	709	4899
04	n**4	34	93

5. Discussion and future directions

While analyzing the time complexity of algorithms it is seen that if the nested loops are used in the application, and the outer loop iterates i times and the inner loop iterates j times, the statements inside the inner loop may be executed a total of i*j times. This is because the inner loop will iterate j times for each of the i iterations of the outer loop. This means that if both the outer and inner loop are dependent on the problem size n, the statements in the inner loop will be executed $O(n^2)$ times. Similarly while considering triply nested loops, all of which are dependent on the problem size n, the statements in the innermost loop will be executed $O(n^3)$ times.

In case of doubly nested loops where only the outer loop is dependent on the problem size n, and the inner loop always executes a constant number of times, the inner loop will execute exactly a constant number of times for each of the n iterations of the outer loop, and so the total number of times the statements in the innermost loop will be executed is O(n) times not $O(n^2)$ times.

6. Conclusion

The algorithms whose solutions are independent of the size of the problem generally may have constant time complexity. The time complexity of an operation could be recognized that usually modifies the data structure.

7. References

[1] E. Horowitz, S. Sahni, S. Rajasekaran, Computer Algorithms/C++, W.H. freeman, 1996.

[2] Dewdney A.K., The new turing omnibus, New York , Henry Holt , 1989.

[3] J.E. Hopcroft, J.D. Ullman, Introduction to automata theory, languages and computation, Addison Wesley 1979.

[4] Graham Cormode and S.Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms, 55(1):58– 75, 2005.

[5] D. Kempe and Frank McSherry. A decentralized algorithm for spectral analaysis. In *Symposium on* Theory of Computing. ACM, 2004.

[6] M uhlenbein¨, H. How genetic algorithms really work : Mutation and hill climbing. In Parallel Problem Solving from Nature, 1992.

[7] Droste, S., Jansen, T., and Wegener, I. (2006). Upper and lower bounds for randomized search heuristics in black-box optimization. Theory of Computing Systems, 2006.

[8] Rylander, B., Foster, J.: Genetic Algorithms, and Hardness, Proceedings of the World Science and Engineering Society's Conference on Soft Computing, 2001.

[9] A. V. Aho and J. D. Ullman. Foundations of Computer Science. W.H. Freeman, 1992.

Authors' Profile

Er. Sambit Kumar Mishra has obtained B.E. and M.Tech. in Computer Science & Engineering from Amaravati University, Maharastra and Indian School of Mines, Dhanbad respectively. He is now serving in ABIT, Cuttack as Associate Professor in the department of Computer Science & Engineering. He has submitted his Ph.D thesis for evaluation. He has 13 number of publications in reputed International Journals. Recently he is also reviewer of peer reviewed international Journals, e.g . International Journal of Scientific and Engineering Research (IJSER) and International Journal of Information Technology and Computer Science(IJITCS), European Journal of Academic Essays.

Er. Pratap Kumar Champati has obtained B.E. in Computer Sc.& Engg. from Utkal University and M.Tech. in Computer Sc.& Engg. from B.P.U.T. Odisha. He is now serving as Sr. Lecturer in ABIT, Cuttack. He has several publications in reputed National , International conferences and Journals.

Er. Rajesh Kumar Sahoo has obtained B.E. in Computer Sc.& Engg. from Utkal University and M.Tech. in Computer Sc.& Engg. from K.I.I.T. University, Bhubaneswar, Odisha. He is now serving as Asst.Professor & Head in department of Computer Sc.& Engg. ABIT, Cuttack. He has several publications in reputed National , International conferences and Journals.